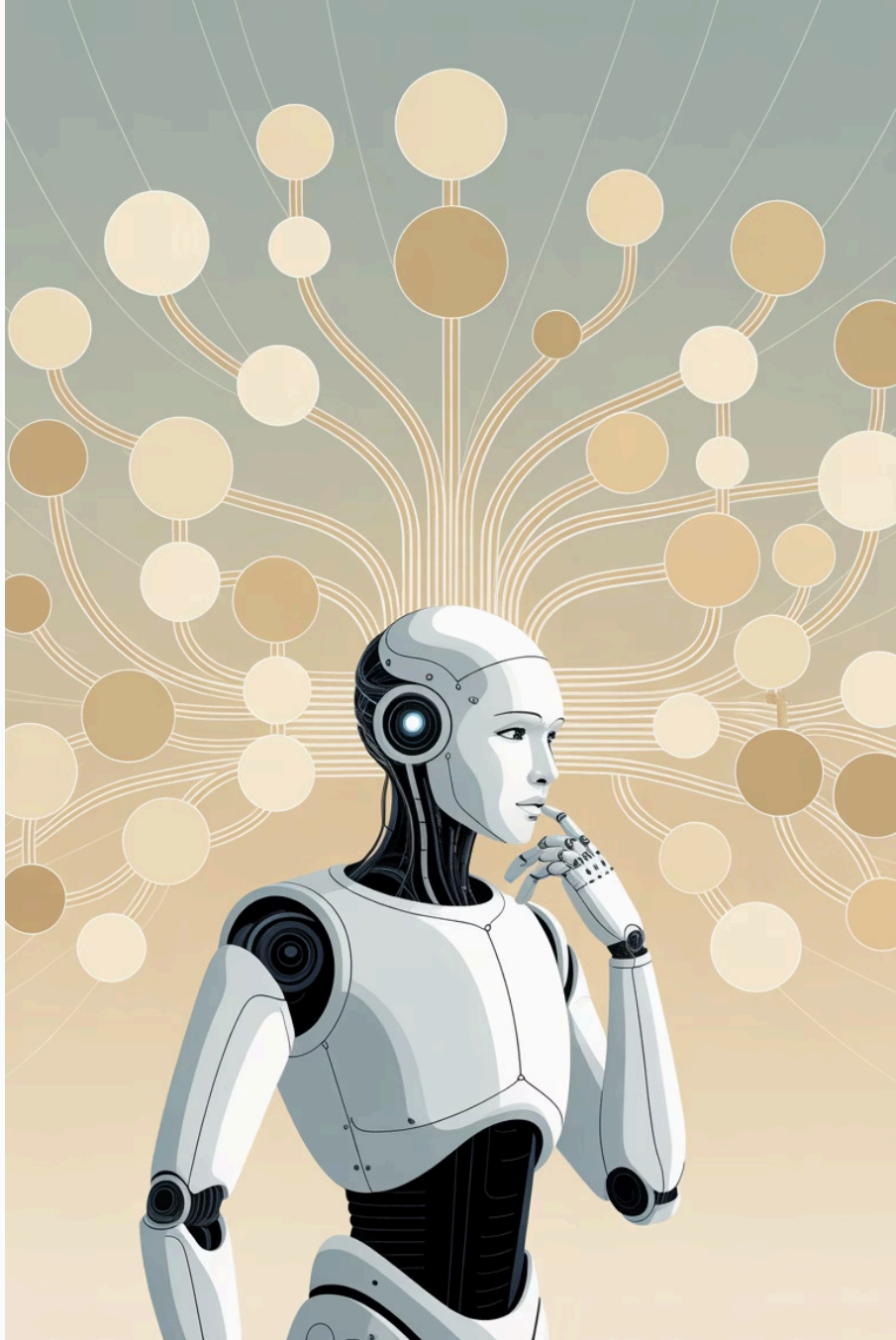
Agentic AI Architecture Guidelines

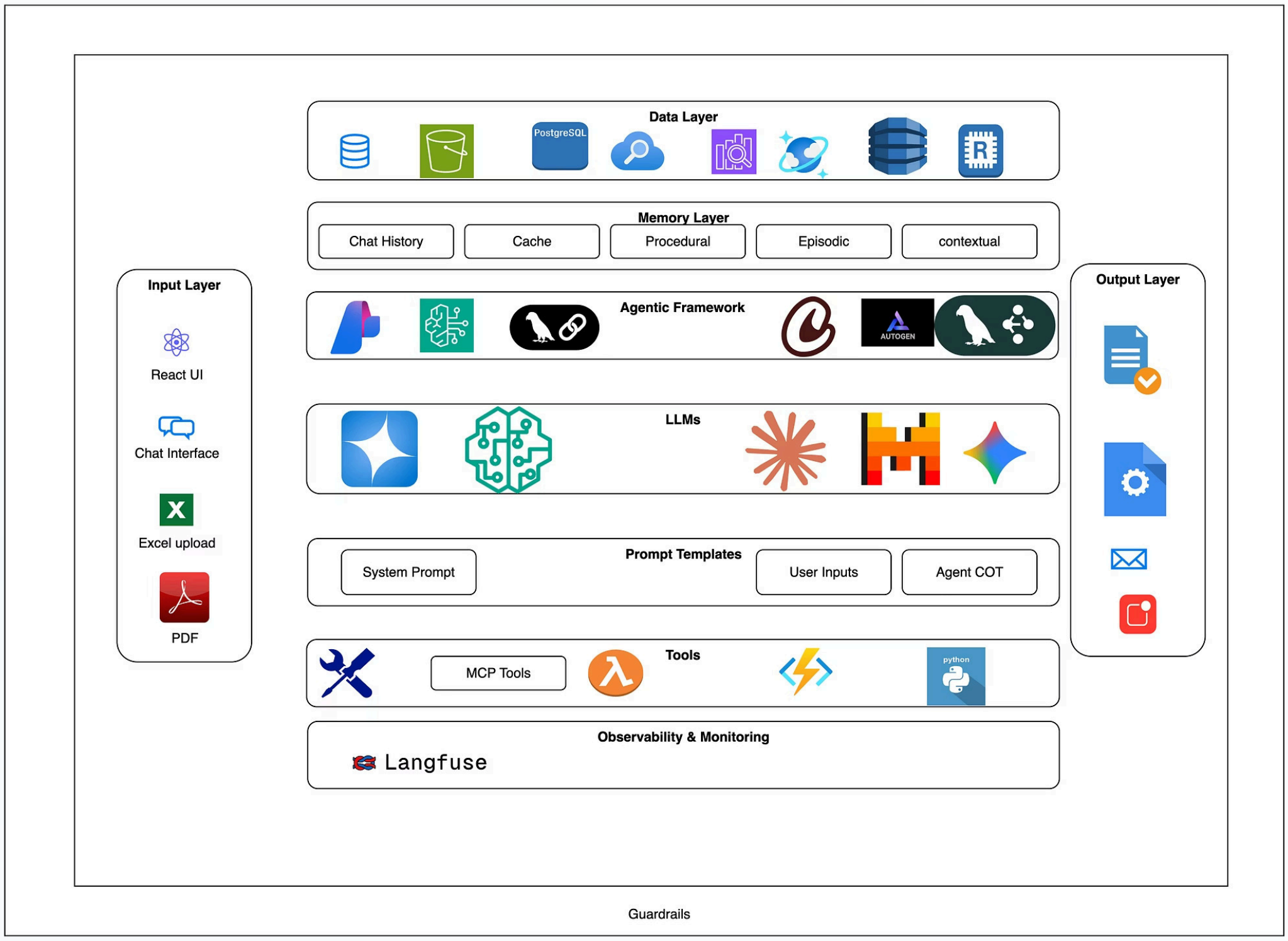
This document provides a comprehensive overview of the foundational reference architecture for Agentic AI systems. The focus is explaining the SaaS developers how they can basically relate this and learn the Agent Architectures & solutions. The key thing to remember is the non deterministic behaviour of LLMs makes the agents to be monitored more aggressively than it is needed for Saas Apps.

Specifically, this guide will explore the core components, various architectural types, and the layered workflow involved in building robust and scalable Agentic AI solutions. We will also examine key design patterns and multi-agent collaboration strategies essential for developing sophisticated, self-improving AI applications.



Agentic AI Systems Reference Architecture

Agentic AI systems are designed to autonomously perceive, plan, decide, and act with minimal human input, moving beyond simple reactive responses. To simplify Agentic AI systems for traditional SaaS developers, consider the UI as the input layer through which agents receive instructions, the backend as the Agentic framework and tool components, and the data layer (memory components) as the foundation of understanding. On top of this foundation, we build the Agentic AI components using prompts, the Agentic framework, LLMs, MCP, A2A, etc. The purpose of this diagram is to show that, for an agent to operate effectively, all these layers are required.



Agentic AI Systems (Simplified for SaaS Developers)

Agentic AI systems are designed to **autonomously perceive, plan, decide, and act** with minimal human input, going beyond simple reactive responses. To map this into a traditional SaaS architecture analogy:

- **UI (Input Layer):** User interactions, tasks, or goals are captured here. This is the entry point for feeding intent into the agents.
- **Backend (Core Agentic Framework and functions):** The orchestration logic that includes agent controllers, planning engines, and decision-making flows.
- **Tools (Action Layer):** Accessed and governed via **MCP (Model Context Protocol)**, which acts as the **tool orchestration protocol**, ensuring secure, authorized, and context-aware tool usage.
- **Data / Memory (Knowledge Layer):** Long-term and short-term memory, embeddings, and contextual understanding that allow agents to learn from history and adapt.

On top of this foundation, we integrate:

- **LLMs:** For reasoning, natural language understanding, and generation.
- **Prompts & Prompt Chains:** To guide the agent's reasoning steps.
- **Agentic Framework Components:** Planning, reflection, error recovery, and multi-step workflows.
- **A2A (Agent-to-Agent Collaboration):** Coordination between multiple specialized agents.

The **Agent Output** is not just a single response but can be an **automated process, structured document, decision log, or communication artifact**—all aligned with the defined business scope.

Core Components of Agentic Architecture

Perception

Gathering and interpreting data from environment or inputs through sensors, natural language understanding, or data pipelines.

Cognition

Planning, decision-making, and goal-setting processes powered by language models or specialized reasoning engines.

Action

Executing decisions via APIs, tools, database queries, or virtual/physical actuators that affect the environment.

Memory

Retaining context and learning from past interactions to continuously improve performance over time.

Types of Agentic Architecture Patterns

Single-Agent Systems

One AI agent handles the entire workflow end-to-end.

- Simpler implementation and control flow
- Well-suited for linear or focused tasks
- Lower computational overhead
- Examples: Personal assistants, specialized tools

Multi-Agent Systems

Multiple specialized agents collaborate to solve complex problems.

- More robust with distributed intelligence
- Supports parallelism and complex workflows
- Enables specialized expertise and coordination
- Examples: Enterprise workflows, simulation environments

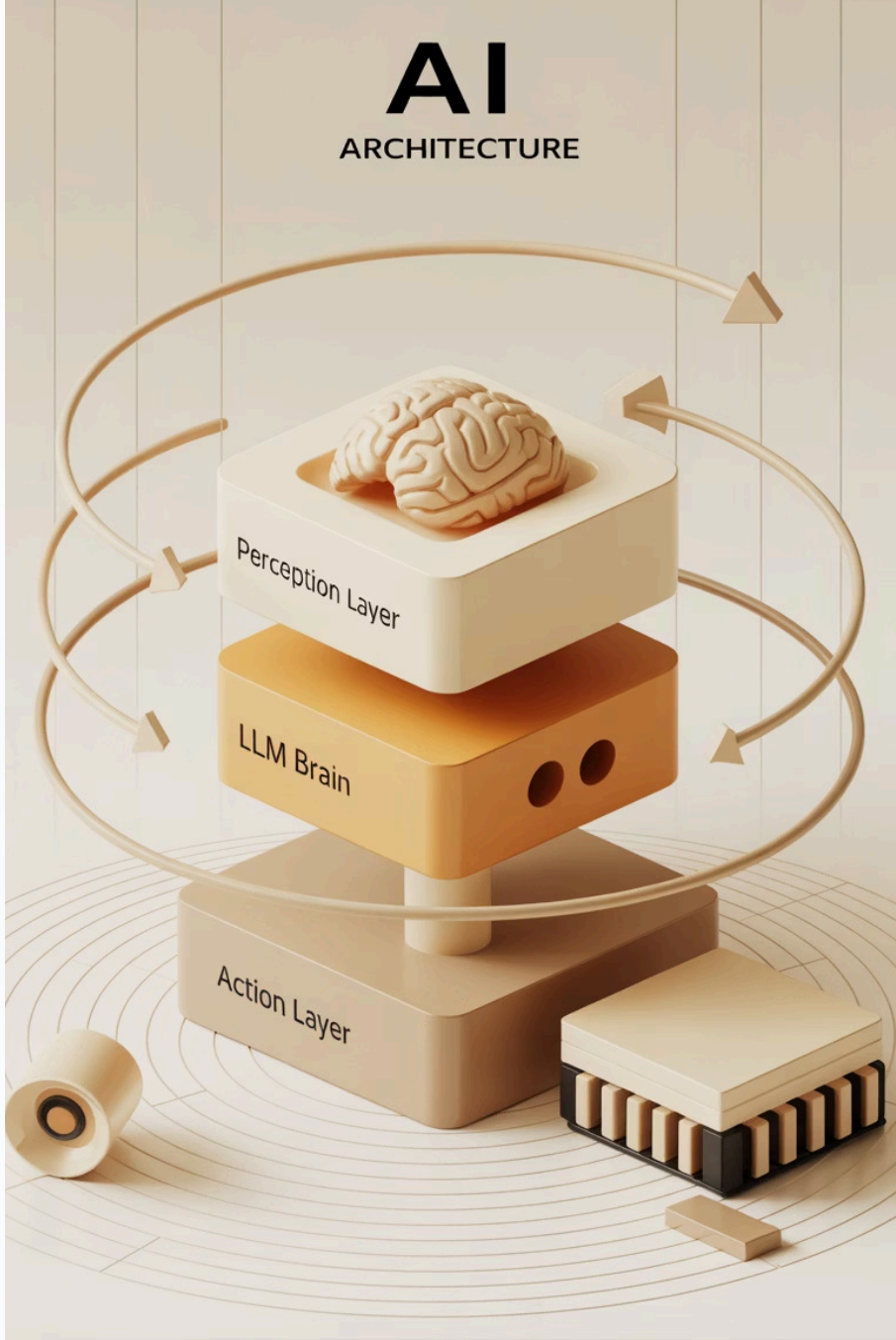
Agentic Workflow patterns

- **Workflows** are systems where LLMs and tools are orchestrated through predefined code paths.
  - **Prompt chaining**
  - **Routing**
  - **Parallelization**
  - **Orchestrator-workers**
  - **Evaluator-optimizer**

Simplified Agentic Workflow

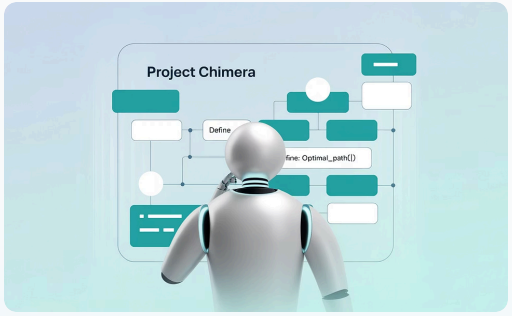
The diagram illustrates how information flows through an agentic system:

1. External data enters through the **Perception Layer**, where it's processed into a format the agent can understand
2. The **Cognition Layer** evaluates the information, sets goals, and makes decisions
3. These decisions activate the **Action Layer**, which interfaces with external systems
4. Results feed back into memory to improve future performance



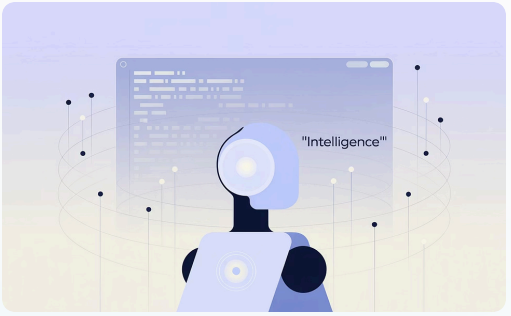
Iterative Agent Development Cycle

Building effective agents is an iterative process, much like traditional software development. The Anthropic team proposes a "PEAR" (Plan, Execute, Analyze, Refine) cycle to guide this process:



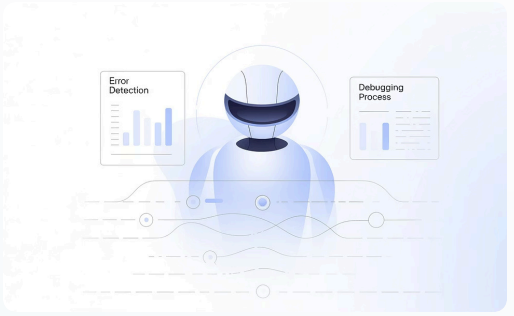
Plan

Define the agent's goal, available tools, and expected output. Outline the desired sequence of steps the agent should take.



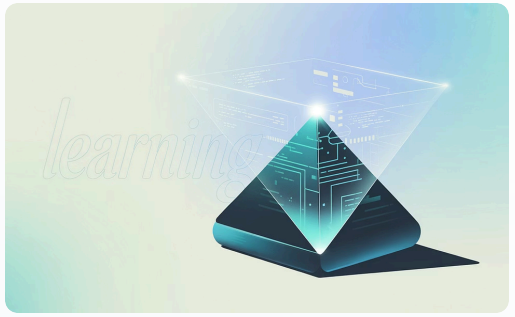
Execute

Run the agent with carefully selected inputs and observe its behavior. This involves testing with real-world or representative data.



Analyze

Evaluate the agent's performance against the plan. Identify deviations, errors, and areas for improvement. Review its thought process and outputs.



Refine

Adjust the agent's prompts, tools, or underlying logic based on the analysis. This could involve prompt engineering, adding new tools, or refining existing ones.

Conclusion

Agentic AI is not just about building autonomous workflows—it's about **designing scalable, resilient, and observable systems** that SaaS developers can adopt without reinventing the wheel. By understanding the parallels between cloud-native principles (like Kubernetes clusters, namespaces, pods, and observability) and Agent architectures, developers can map their existing SaaS skills directly into this new paradigm.

The journey ahead will demand not only technical proficiency but also **new governance models, interoperability standards, and multi-agent collaboration patterns**. For SaaS builders, this is an opportunity to move beyond traditional app development and embrace **self-adaptive systems that continuously learn, adapt, and optimize**.

As you explore Agentic AI architectures, remember:

- **Think cloud-native** → Agents scale like microservices.
- **Design for observability** → What you cannot measure, you cannot govern.
- **Leverage collaboration** → Multi-agent systems thrive when designed to communicate effectively.
- **Keep it modular** → Just like SaaS, Agents benefit from clear boundaries and reusable components.

In short, **the same discipline that shaped SaaS scalability can now shape Agentic AI systems.** Developers who embrace this shift will not just build applications—they will build the future of intelligent, adaptive software.